Diagnosability verication with Petri net unfoldings

Agnes Madalinski, Farid Nouioua, Philippe Dague

Diagnosability verification with Petri net unfoldings

Agnes Madalinski[1⋆], Farid Nouioua[2⋆] and Philippe Dague[3]

[1]Faculty of Engineering Science, Univ. Austral de Chile, Valdivia, Chile
[2]Laboratoire des sciences de l'information et des systèmes, Marseille, France
[3]LRI, Univ. Paris-Sud, CNRS/INRIA Saclay, Orsay, France

Abstract Complex systems increasingly require safety and robustness w.r.t faults occurrences, and diagnosability is a key property to ensure this at design stage. This paper demonstrates how Petri net unfoldings, which have been proven to elevate the state explosion problem, can be applied to verify diagnosability by adapting the *twin plant* method.

1 Introduction

Studying *diagnosability* of systems represents an important domain which has drawn in the last years the attention of many researchers in both artificial intelligence and control theory communities. Diagnosability is an important property that determines the ability of a system to detect faults occurrences given only observable sequences (the system has observable events and unobservable events including faults). If a system is diagnosable the diagnosis will find an accurate explanation for any possible set of observations from the system, otherwise the diagnosis will give an ambiguous and useless explanation.

The seminal work in [12] has introduced a formal framework for analysing the diagnosability properties of discrete event systems (DES) represented by finite automata. The proposed method for diagnosability verification is based on the construction of a *diagnoser*: an automaton with only observable events which allows one to estimate states of the system after observation of sequences. Improvements based on the *twin plant* method has been proposed in the centralised framework [5,14] and in the distributed one [13]. The basic idea is to build a *verifier* from a diagnoser by constructing the synchronous product of the diagnoser with itself on observable events. The verifier compares every pair of paths in the system that have the same observable behaviour. [13] uses the modularity of the system to compute local twin plants and to test diagnosability by gradually combining local twin plants (in the worst case building the global twin plant). Other interesting advances in this domain include the use of model checking [11] and process algebra [1] to test diagnosability, and the adaptation of the diagnosability definition to deal with enriched models such as stochastic automata [10].

Most of the previous approaches operate on variants of finite state machine-based models. Naturally, these models suffer from the state space explosion problem. To alleviate this problem Petri net (PN) unfolding techniques appear

⋆ This work was done while the authors performed a post-doc at LRI, Orsay, France.

promising. A finite and complete prefix of a PN unfolding [2,7] gives a compact representation of all behaviours and reachable markings of this PN in a partial order. Executions are considered as partial ordered set of events rather than sequences, which results in memory savings. The unfolding prefix has been used in various applications (see e.g. [6]) such as distributed diagnosis, model checking, synthesis of asynchronous circuits or planning problems. Also diagnosability has been studied in this context from a purely theoretical point of view: [4] proposes a definition of diagnosability based on observable partial orders and, opposed to such quantitative criteria, a qualitative notion specific to partial orders has been introduced in [3]. The main difference between their definition and that proposed in this paper lies on the granularity level of observations. In the definition proposed in [4] any execution of a partial order corresponds to the same observation, whereas in this work the different executions of a partial order correspond to different observations.

The objective of this paper is to use PN unfolding prefixes to verify diagnosability by adapting the twin plant method [14]; with the long term objective to develop an approach to verify diagnosability in a distributed way in the framework of modular complete prefixes [8]. The used model is a labelled Petri net, where transitions are labelled with observable and unobservable events. Diagnosability is tested using a finite and complete prefix of a verifier. The verifier is obtained by the synchronous product of a diagnoser (the system enriched with information about the occurrence of faults). A necessary and sufficient condition for diagnosability is given. In addition, two algorithms are given to test diagnosability; moreover, two improvements are presented, which exploit the symmetry and "interesting' behaviour of the verifier to reduce its size. An extended version of this paper in form of a technical report, with the algorithms and the detailed proofs of lemmas 1, 2 and proposition 1, can be found in [9].

2 Petri nets and their unfoldings

Petri nets A *Petri net* is a quadruple $N = (P, T, \rightarrow, M^0)$ such that P and T are disjoint sets of *places* and *transitions*, respectively, $\rightarrow \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*, and M^0 is the *initial marking*, where a marking is a function $P \rightarrow \mathbb{N} = \{0, 1, 2, ...\}$ which assigns a number of *tokens* to each place. A transition t is *enabled* under a marking M (denoted by $M[t\rangle$) and yields M' when firing (denoted by $M[t\rangle M'$). A transitions sequence $\sigma = t_1, ..., t_k \in T$ is a *firing sequence* from M_1 to M_{k+1}, denoted by $M_1[\sigma\rangle M_{k+1}$ or $M_1[\sigma\rangle$, iff a set of markings $M_2, ..., M_{k+1}$ exit such that $M_i[t_i\rangle M_{i+1}$, $1 \leq i \leq k$. A net N is *safe* if for every reachable marking M and every place $p \in P$, $M(p) \subseteq \{0, 1\}$.

Canonical prefixes A *finite and complete unfolding prefix* $Pref_N$ of N is a finite acyclic net which implicitly represents all the reachable states of N together with transitions enabled at those states. Intuitively, it can be obtained through *unfolding* N, by successive firings of transitions, under the following assumptions: (a) for each new firing a fresh transition (called an *event*) is generated; (b) for each newly produced token a fresh place (called a *condition*) is generated.

Due to its structural properties (such as acyclicity) the reachable markings of N can be represented using *configurations* of $Pref_N$. A configuration κ is a

downward-closed set of events (it means that if $e \in \kappa$ and f is a causal predecessor of e, then $f \in \kappa$) without structural conflicts . Intuitively, a configuration is a partial-order execution, i.e. an execution where the order of firing of concurrent events is not important. The *basic* configuration of an event e, denoted by $[e]$, is the smallest (w.r.t set inclusion) configuration containing e (it consists of e and its causal predecessors); $Mark(\kappa)$ denotes the corresponding marking of N, reached by firing a transition sequence corresponding to the events in κ.

The unfolding is infinite whenever N has an infinite run; however, if N has finitely many reachable states then the unfolding eventually starts to repeat itself and can be truncated (by identifying a set of cut-off events) without loss of information, yielding a finite and complete prefix. Intuitively, an event e is a cut-off event if the already build part of the prefix contains an event f (called the corresponding event of e) such that $Mark([f]) = Mark([e])$, where $[f]$ is smaller than $[e]$ w.r.t. some well-founded partial order on configurations of the unfolding called an *adequate order* [2]. We denote by C, E and $Ecut$ the sets of conditions, events and cut-off events of the prefix, respectively, and by $h : E \cup C \to T \cup P$ the mapping from the nodes of the prefix to the corresponding notes of N.

3 Diagnosability

System model The system is modelled with a safe *labelled* Petri net $\mathcal{N} = (N, O, U, \ell)$, which is a Petri net N extended with sets of *observable* and *unobservable* transition labels O and U, respectively, and a labelling function $\ell : T \to O \cup U$ on transitions. The observable transitions correspond to controller commands, sensor readings and their changes, and in contrast, unobservable transitions correspond to some internal events that cause changes in the system not recorded by sensors. The set of *fault* transition labels $F \subseteq O \cup U$ and it is assumed that $F \subseteq U$ since it is trivial to diagnose fault transitions that are observable. Moreover, $F = F_1 \cup ... \cup F_n$ is partitioned into disjoint sets, where F_i denotes the set of fault transitions corresponding to a fault type i such that $1 \leq i \leq n$ and n is the number of fault types. This allows one to handle subsets of faults if it is not necessary to detect uniquely every fault transition. The labelled Petri net $\mathcal{N}$ inherits the operational semantics of the underlying net N. One has $M\,[\ell(t)\rangle\,M'$ if $M\,[t\rangle\,M'$. Moreover, a firing sequence $\sigma \in O \cup U$ is called a *trace* of $\mathcal{N}$ if $M\,[\sigma\rangle$. An example of a system is illustrated in Figure 1(a) with highlighted set of observable transitions labelled with $O = \{a, b, c\}$, and the set of unobservable transitions labelled with $U = \{u, f_1, f_2\}$ including $F = F_1 \cup F_2$, where $F_1 = \{f_1\}$ and $F_2 = \{f_2\}$.

Informally, a system is *diagnosable* if an occurrence of a fault can be detected with certainty in a bounded time. For a system with finite state space this can be expressed as the absence of two infinite traces having the same observable transitions, where one of them contains a failure and the other one does not (see [9] for a formal definition).

Diagnoser In the diagnoser only observable transition are visible and unobservable transitions can be seen as silent transitions. Thus, in order to keep track of the occurrences of faults an additional piece of information is needed. Each marking M of $\mathcal{N}$ is associated with a *fault label function* $\nu : F \to \{0, 1\}$. The

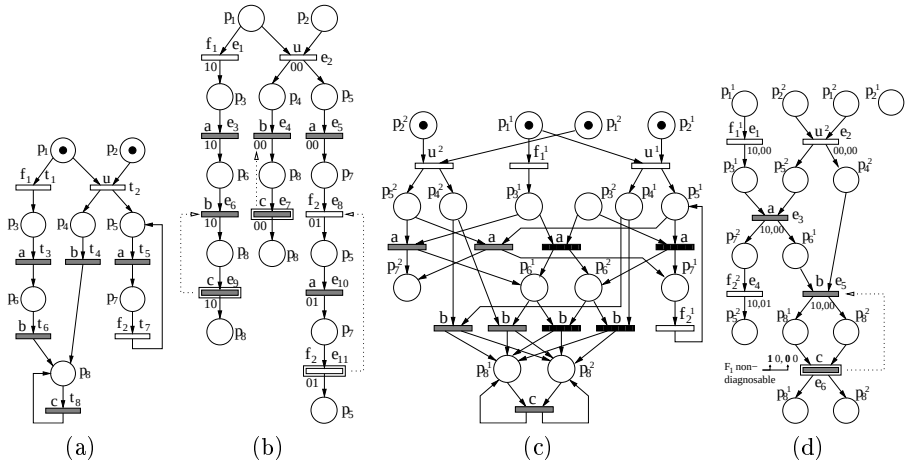


Figure 1: The system $\mathcal{N}$ (a), the canonical prefix of the diagnoser D (b) Petri net of the verifier V (c) and part of its prefix showing non-diagnosable traces (d).

initial marking M^0 is the zero vector; each marking M such that $M^0\,[\sigma\rangle\,M$ is associated with ν as follows: if a fault type occurred in σ then its label is set to $'1'$ and otherwise to $'0'$. Hence, a *diagnoser state* of $\mathcal{N}$ is a pair (M,ν), where M is a marking of $\mathcal{N}$ and ν is its fault label. Let $Fault((M,\nu)) \stackrel{\text{df}}{=} \nu$. The diagnoser can be represented as $D = (\mathcal{N},\nu^0)$ with $(M,\nu)\,[\ell(t)\rangle\,(M',\nu')$ if $M\,[\ell(t)\rangle\,M'$ and ν,ν' are the fault labels associated to M and M', respectively. Note that the number of states of a diagnoser grows by 2^n compared with $\mathcal{N}$.

A canonical prefix of $\mathcal{N}$ is $Pref_N$ augmented with additional labelling of its events $\ell\circ h : E \to O\cup U$. The function "Fault" is extended to configuration of the unfolding of D with $Fault(\kappa) \stackrel{\text{df}}{=} Fault(Mark(\kappa))$. Figure 1 (a) and (b) depicts the diagnoser D and its canonical prefix $Pref_D^{\Theta^{tot}}$ with the total adequate order [2] (in figures the cut-off events are drawn as double boxes, and dotted lines indicate the corresponding events of cut-off events; in addition, to each event e is associated a fault vector of $[e]$, which is shown next to events). Observe that in a canonical prefix of $\mathcal{N}$ the event e_8 would be a cut-off event with the corresponding event e_2 since only the relation on marking is considered. However, their codes are different and therefore e_8 cannot be designated as a cut-off event of a diagnoser; the prefix has to be extended until the cut-off event e_{11}, which has a corresponding event e_8.

Synchronous product The synchronous product $\mathcal{N} = \mathcal{N}_1 \times \mathcal{N}_2$ of two labelled Petri nets $\mathcal{N}_1$ and $\mathcal{N}_2$ is defined on common observable events. Intuitively, the product merges transitions having a common observable label and preserves the remaining transitions and all places of each components (see [9] for formal definition). Figure 1(c) shows the product of the running example with itself (a superscript is used to indicate the component of the non-synchronised nodes). Observe that there are four transitions labelled with a (and also with b). This is due to

the fact that t_3 and t_5 are labelled with a in the original net, and in the product they synchronise to $(t_3, t_3), (t_3, t_5), (t_5, t_3)$ and (t_5, t_5) (and similar applies to transitions labelled with b). Note that the firing $(M_1, M_2)\,[\ell\,((t_3, t_3))\rangle\,(M_1', M_2')$ of $\mathcal{N}$ corresponds to firings $M_1\,[\ell_1\,(t_3)\rangle\,M_1'$ in $\mathcal{N}_1$ and $M_2\,[\ell_2\,(t_3)\rangle\,M_2'$ in $\mathcal{N}_2$.

Verifier In order to check the diagnosability property a verifier is build from the synchronous product of the diagnoser with itself. The *verifier* $V = D \times D = \left(\mathcal{N} \times \mathcal{N}, \left(\nu^0, \nu^0\right)\right)$, where $D = (\mathcal{N}, \nu^0)$ is a diagnoser, $\mathcal{N}$ is the system model and $\left(\nu^0, \nu^0\right)$ is the synchronised fault label. In the sequel V is denoted as $\left(\mathcal{N}, \mathcal{N}, \nu^0, \nu^0\right)$ for the sake of simplicity. Thus, V can be regarded as a diagnoser net (it has the same dynamics as a diagnoser net). The canonical prefix of the verifier V, $Pref_V^{\subset}$ with the adequate order $\subset$.

Condition for diagnosability A trace σ in V forms a *cycle* if there exists a trace σ' within σ such that $\left(M^1, M^2, \nu^1, \nu^2\right)\,[\sigma'\rangle\,\left(M^1, M^2, \nu^1, \nu^2\right)$ and $\sigma' \neq \emptyset$. Let $e \in Ecut_V$ be a cut-off event in $Pref_V^{\subset}$ then $\kappa^e = \{\kappa : [e] \subseteq \kappa\}$.

Lemma 1. *The following holds.*

1. *for each cycle in the verifier V there is a configuration $\kappa \in \kappa^e$ s. t. $e \in Ecut_V$*
2. *for each $\kappa \in \kappa^e$, where $e \in Ecut_V$, there exists a cycle in the verifier V*

It is assumed that there are no cycles of unobservable events; however, if this assumption is dropped it can be easily checked while building the canonical prefix (by checking whether an observable event occurs between the cut-off events and their corresponding events; note that the corresponding event of a cut-off event is always in the history of its cut-off event since $\subset$ is used as adequate order).

Lemma 2. *Let $q = \left(M^1, M^2, \nu^1, \nu^2\right)$ and $q' = \left(M'^1, M'^2, \nu'^1, \nu'^2\right)$ be two states in a cycle in V then $\nu^1 = \nu'^1$ and $\nu^2 = \nu'^2$.*

The verifier compares every pair of traces which have the same observables. Thus, if a cut-off event e occurs this means that there exists a cycle (between e and its corresponding event). Furthermore, if there exist different local fault labels of F_i, i.e. $\nu_i^1 \neq \nu_i^2$ with $Fault([e]) = \left(\nu^1, \nu^2\right)$, the system is not diagnosable w.r.t. F_i. This is illustrated in Figure 1(d), where a part of $Pref_V^{\subset}$ is shown. The cut-off event e_6 and its corresponding event e_5 form a cycle since their basic configurations reach the same marking $\left\{p_2^1, p_8^1, p_7^2, p_8^2\right\}$. It is evident that the system is not diagnosable w.r.t. F_1 since their corresponding fault labels are different. Thus, there exist two equivalent observable traces in the system one with an occurrence of F_1 (f_1^1, a, b, c) and the other without any occurrence of F_1 (u^2, a, b, c). They can never be distinguished since they are within a cycle.

However, it is not enough to check the basic configuration of a cut-off event e since its concurrent events can influence the decision. This is also illustrated in Figure 1(d) with the fault type F_2. The fault label of $[e_6]$ indicates that there is no ambiguity, yet the occurrence of the concurrent event e_4, which corresponds to F_2, changes one of the fault label of F_2 and the system becomes not diagnosable w.r.t. F_2. One has f_1^1, a, b, c and u^2, a, b, c with the occurrence of f_2^2.

Proposition 1. *Let $Pref_V^{\subset}$ be the canonical prefix of the verifier V with its set of cut-off events $Ecut_V$. Then V is called* F_i-diagnosable *w.r.t. O and F_i if $\forall e \in Ecut_V\ \forall \kappa \in \kappa^e, \nu_i^1 = \nu_i^2$, where $Fault(\kappa) = \left(\nu^1, \nu^2\right)$. Moreover, the system $\mathcal{N}$ is* diagnosable *w.r.t. O and F if it is F_i-diagnosable for all $F_i \in F$.*

4 Verification of diagnosability

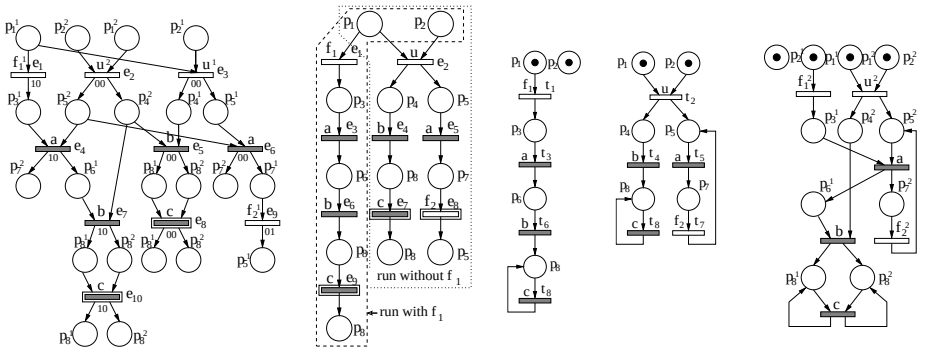
The canonical prefix $Pref_V^{\subset}$ can be built by using the algorithm presented in [7]. To test diagnosability it suffices to examine the cut-off events and their concurrent events for ambiguous fault labels. For a non-diagnosable system a set of configurations can be extracted from $Pref_V^{\subset}$ to show ambiguous explanations.

A depth-first approach as opposed to the breadth-first one can be also employed to test diagnosability. It is advantageous if simply an answer about the system's diagnosability is needed. The depth-first search tries to extend a branch of a prefix before considering other branches. To do that the diagnosability check has to be extended, not only cut-off events are candidates but also their concurrent events are. (See [9] for algorithms).

To reduce complexity of the verifier one can consider one fault type at a time and perform the diagnosability check n times w.r.t. to the fault type set $F_1, ..., F_n$ as in [5] (by setting other faults as non-fault unobservables). The complexity is then linear in the number of faults (the state space reduces by 2^{n-1}).

Contracted verifier It would be advantageous to exploit the symmetry of the verifier. Recall that the verifier compares every pair of equivalent observable traces corresponding to the diagnosers, D^1 and D^2. There are four cases: (1) a fault $f^{\mathbf{1}}$ occurs in D^1 but its counterpart $f^{\mathbf{2}}$ does not occur in D^2 and (2) visa verse (showing that the system is not diagnosable if occurring in a infinite trace), and (3) the occurrence of both $f^{\mathbf{1}}$ and $f^{\mathbf{2}}$ and (4) the inverse case (indicating that the considered trace is diagnosable). Due to the symmetry it is sufficient to consider either Case 1 or 2 (e.g. let consider Case 1). Moreover, the Case 4 can be made redundant by removing from the verifier V the fault transitions F^2 corresponding to the diagnoser D^2 together with their arcs resulting in a *contracted* verifier V_c. By doing this the traces containing a fault of F^2 are not reachable leaving only Case 1 and 3. Thus, the diagnosability check is reduced: the system is not diagnosable if there exists an infinite trace containing a fault corresponding to D^1. The verifier in Figure 1(c) becomes a contracted one if the fault transitions $F^2 = \{f_1^2, f_2^2\}$ and their arcs are removed. The transition marked black correspond to transitions which are not reachable due to the removal. This is evident on the canonical prefix depicted in Figure 2(a). Note that the fault vector corresponding to D^1 is only important for the diagnosability check since the other fault vectors are always zero. There are two cut-off events, e_8 and e_{10}; it is immediately evident from the fault vector of e_{10} that the system is not diagnosable w.r.t. F_1, and it is later evident that there exist a configuration in κ^{e_8} (containing e_8 and its concurrent event e_9, which correspond to f_2^1) that shows that the system is also not diagnosable w.r.t. F_2.

Reduced verifier w.r.t a fault One can go a step further and consider a verifier built out of the product of a diagnoser containing only fault occurrences and a diagnoser containing only non-fault occurrences. This is especially interesting for the case where one fault type at a time is considered due to the size reduction. Given a diagnoser D in which one fault type F_i is considered (i.e. $F \setminus F_i$ is set to non-fault unobservables) one has w.r.t. the fault f_i the *reduced* verifier $V_{f_i} = D_{f_i} \times D_{\bar{f}_i}$, where f_i is any fault in F_i, $1 \leq i \leq n$, and the reduced



(a) $Pref^{\subset}_{V_c}$ (b) $Pref^{\subset}_{\mathcal{N}}$ (c) D_{f_1} (d) $D_{\bar{f}_1}$ (e) V_{f_1}=D_{f_1}×$D_{\bar{f}_1}$

Figure 2: Reducing the verifier's complexity.

diagnoser D_{f_i} ($D_{\bar{f}_i}$), which corresponds to the part of D containing (non-)f_i-fault occurrences. Extracting fault and non-fault occurrences at the level of the diagnoser is not straightforward due to the cyclicity of the net. However, it is possible to extract this information from the canonical prefix of the underlying system of the diagnoser $Pref^{\subset}_{\mathcal{N}}$ (with $\subset$ as adequate order) by examining maximal configurations w.r.t. set inclusion, called *runs*.

Let Ω be the set of runs in $Pref^{\Theta}_{\mathcal{N}}$. Then, $\Omega_{f_i} = \{\omega \in \Omega / \exists e \in \omega : \ell \circ h(e) \in F_i\}$ and $\Omega_{\bar{f}_i} = \{\omega \in \Omega / \forall e \in \omega : \ell \circ h(e) \notin F_i\}$, where e is an event in $Pref^{\Theta}_{\mathcal{N}}$. In other words Ω_{f_i} ($\Omega_{\bar{f}_i}$) is the set of runs where faults from type F_i (not) occur. By definition, Ω_{f_i} ($\Omega_{\bar{f}_i}$) corresponds to all the possible executions of the reduced underlying system model representing the reduced diagnoser D_{f_i} ($D_{\bar{f}_i}$). Hence, the projection of Ω_{f_i} and $\Omega_{\bar{f}_i}$ onto the diagnoser D (via its underlying system model $\mathcal{N}$) correspond to D_{f_i} and $D_{\bar{f}_i}$, respectively. For each fault the reduced verifier is constructed from $Pref^{\subset}_{\mathcal{N}}$, which is unchanged, only the extracted information differs. Thus, it has to be build only once.

The process of obtaining V_{f_1} applied to the running example is illustrated in Figure 2(b)-(e). There are two runs in $Pref^{\subset}_{\mathcal{N}}$, one with the fault f_1 and the other without f_1. From the runs the reduced diagnosers D_{f_1} and $D_{\bar{f}_1}$ are obtained by projecting them on D; ($Pref^{\subset}_{V_{f_1}}$ corresponds to the one in Figure 1(d) with the fault vector corresponding to F_1). Incidentally, the reduced verifier w.r.t. F_2, V_{f_2}, is similar to V_{f_1}, they only differ in the fault vectors; D_{f_2} corresponds to the run in Figure 2(c) and $D_{\bar{f}_2}$ correspond to the run in Figure 2(b).

Complexity A canonical prefix $Pref_V$ can be exponentially smaller than the reachability graph of V, especially if V exhibits a high degree of concurrency combined with a moderate number of branching behaviour. However, in worst case $Pref_V$ can be exponential in the size of V. In spite of that, the proposed improvements offer a size reduction of $Pref_V$. In particular, when taking into account the symmetry of the verifier or considering one fault type at a time the size reduction can be significant when building $Pref_V$ from the contracted

and/or reduced verifier. Moreover, the depth-first approach together with the improvements may offer a more efficient way that the breath-first one.

5 Conclusion

An approach is proposed to verify diagnosability in the framework of PN unfoldings based on the twin plant method. It consists in constructing a verifier, which compares pairs of paths from the initial model sharing the same observable behaviour. In the canonical prefix of the verifier the diagnosability test is reduced to the comparison of binary vector pairs of configurations associated with cut-off events. Each configuration is linked with a pair of binary vectors containing information about fault occurrences in two executions sharing the same observables. This is further reduced in a contracted verifier where only binary vectors of one configuration instead of the pair is examined while considering a reduced reachable space. Moreover, other proposed improvements can be applied to reduce the complexity.

Acknowledgements

Many thanks to Stefan Haar and Victor Khomenko for interesting discussions.

References

1. L. Console, C. Picardi, and M. Ribaudo. Diagnosis and diagnosability analysis using pepa. In *Proceedings 14th European Conference on AI - ECAI00*, 2000.
2. J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. *Form. Methods Syst. Des.*, 20(3):285–310, 2002.
3. S. Haar. Unfold and Cover: Qualitative Diagnosability for Petri Nets. In *Proceedings CDC*, 2007.
4. S. Haar, A. Benveniste, E. Fabre, and C. Jard. Partial Order Diagnosability of Discrete Event Systems using Petri Nets Unfoldings. In *Proceedings CDC*, 2003.
5. S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete event systems. In *IEEE Trans. on Aut. Cont.*, 2001.
6. V. Khomenko and E. Fabre (Eds.). Proceedings of the Workshop on UnFOlding and partial order techniques (UFO). Siedlce, Poland, 2007.
7. V. Khomenko, M. Koutny, and W. Vogler. Canonical prefixes of petri net unfoldings. *Acta Informatica, Volume 40, Number 2*, pages 95–118, 2003.
8. A. Madalinski and E. Fabre. Modular construction of finite and complete prefixes. In *Int. Conf. on Application of Concurrency to System Design*, 2008.
9. A. Madalinski, F. Nouioua, and P. Dague. Diagnosability verification with Petri net unfoldings. *LRI technical report no 1516*, 2009.
10. F. Nouioua and P. Dague. A probabilistic analysis of diagnosability in discrete event systems. In *Proceedings 18th European Conference on AI - ECAI08*, 2008.
11. C. Pecheur, A. Cimatti, and R. Cimatti. Formal verification of diagnosability via symbolic model checking. In *18th International Joint Conference on AI*, 2003.
12. M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of Discrete Events Systems. *IEEE Trans. on Aut. Cont*, 1995.
13. A. Schumann and Y. Pencolé. Scalable diagnosability checking of event-driven systems. In *20th International Joint Conference on AI*, 2007.
14. Yoo T. and S. Lafortune. Polynomial-Time Verification of Diagnosability of Partially Observed Discrete-Event Systems. *IEEE Trans. on Aut. Cont.*, 2002.